# 03 - Intro to graphics (with ggplot2)

*ST 597 | Spring 2017*
*University of Alabama*

*03-dataviz.pdf*

## Contents

**1 Intro to R Graphics** **2**
- 1.1 Graphics Packages . . . 2
- 1.2 Base Graphics . . . 2
- 1.3 `plot()` . . . 2
- 1.4 ggplot2 package . . . 2

**2 Scatterplots** **3**
- 2.1 `heightweight` data . . . 3
- 2.2 Data Frames (and Tibbles) . . . 3
- 2.3 Basic Scatterplot . . . 4
- 2.4 Aesthetics . . . 5
- 2.5 Your Turn: Scatterplots . . . 9
- 2.6 Additional Geoms . . . 9
- 2.7 Layers . . . 10
- 2.8 Your Turn: Geoms and Layers . . . 13
- 2.9 Plot Objects . . . 13
- 2.10 Scatterplot Aesthetics . . . 14

**3 Bar Graphs: `geom_bar()`** **15**
- 3.1 `diamonds` data . . . 15
- 3.2 Bar graphs . . . 15
- 3.3 `geom_bar()` . . . 15
- 3.4 Two Variables . . . 16
- 3.5 Stats: `stat_count()` and `stat_identity()` . . . 18
- 3.6 Reordering x-axis `reorder()` . . . 21
- 3.7 Your Turn: Bar Graphs . . . 22

**4 Additional Material** **22**
- 4.1 ggplot 2 details . . . 22
- 4.2 Themes . . . 23
- 4.3 Scales . . . 23

---

### Required Packages and Data

```r
library(tidyverse)
library(gcookbook)
```

# 1 Intro to R Graphics

## 1.1 Graphics Packages

R has several approaches to making graphics:

1. **Base Graphics** - the golden oldies. Includes functions like `plot()`, `lines()`, `points()`, `barplot()`, `boxplot()`, `hist()` etc.
   - Graphics are layered manually. First create high level plots (e.g, with `plot`), then add on top with e.g., `lines()` or `text()`
2. **ggplot2** - Grammar of Graphics created by Hadley Wickham.
3. **lattice** - a popular approach, but we will not cover in this course.

## 1.2 Base Graphics

Calling a *high-level* plotting function creates a new plot.

- `barplot()`, `boxplot()`, `curve()`, `hist()`, `plot()`, `dotchart()`, `image()`, `matplot()`, `mosaicplot()`, `stripchart()`, `contour()`

*Low-level* functions write on top of the existing plot.

- Add to the plotting region: `abline()`, `lines()`, `segments()`, `points()`, `polygon()`, `grid()`
- Add text: `legend()`, `text()`, `mtext()`
- Modify/add axes: `axis()`, `box()`, `rug()`, `title()`

## 1.3 `plot()`

The `plot(x)` function can produce plots depending on the class of object x

- if x is data.frame, then a `pairs()` plot
- if x is a factor vector, then a `barplot()`
- if x is a linear model (`lm()`), then a series of regression diagnostic plots
- Or, we have been creating scatterplots with `plot(x,y)`

Advanced: type `methods(plot)` to see all the types of objects that `plot()` knows about. Some packages add their own plotting methods that can be called with `plot()`. To see help documentation, type in the full method (e.g., `?plot.data.frame`). To see the code that is used (for the methods with asterisks) use the `getAnywhere()` function, e.g. `getAnywhere(plot.data.frame)`.

## 1.4 ggplot2 package

The `ggplot2` package is created by Hadley Wickham and is the 2nd version of a *grammar of graphics* approach to visualizing data. It takes a somewhat different approach than the base R graphics, which we will illustrate with some examples. There are now several nice resources available:

1. Data Visualization Cheat Sheet
2. ggplot2 website
3. R Graphics Cookbook, by Winston Chang

    - Associated website
4. ggplot2 Theory

## 2 Scatterplots

### 2.1 `heightweight` data

Check out the `heightweight` data from the `gcookbook` package (`?heightweight`). It is a sample of 236 schoolchildren.

```
library(gcookbook)  # to access the heightweight data
data(heightweight)
str(heightweight)
#> 'data.frame':    236 obs. of  5 variables:
#>  $ sex     : Factor w/ 2 levels "f","m": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ ageYear : num  11.9 12.9 12.8 13.4 15.9 ...
#>  $ ageMonth: int  143 155 153 161 191 171 185 142 160 140 ...
#>  $ heightIn: num  56.3 62.3 63.3 59 62.5 62.5 59 56.5 62 53.8 ...
#>  $ weightLb: num  85 105 108 92 112 ...
```

### 2.2 Data Frames (and Tibbles)

A `data.frame` (and `tibble`) is similar to a spreadsheet or data table: data represented in rows and columns.

- Technically, we can think of a data frame as a collection of `vectors` that **all have the same length**.
  - $n$ rows/observations, $p$ columns/variables/features
- But they don't have to be of the same *type*. E.g., some columns are character vectors, some numeric vectors, some factors, etc.

Think of each row of the data frame as an observation and each column as a variable.

#### 2.2.1 Getting info about a data frame

- Some useful functions

```
ncol(heightweight)    # ncol() gives number of columns
#> [1] 5
nrow(heightweight)    # nrow() gives number of rows
#> [1] 236
dim(heightweight)     # dim() gives dimensions (nrows, ncols)
#> [1] 236   5
```

- The full data frame can be viewed with the function `View()` (capital V)

```
View(heightweight)
```

- The function `str()` will give information about a data frame (or any other R object)

```
str(heightweight)
#> 'data.frame':    236 obs. of  5 variables:
#>  $ sex     : Factor w/ 2 levels "f","m": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ ageYear : num  11.9 12.9 12.8 13.4 15.9 ...
#>  $ ageMonth: int  143 155 153 161 191 171 185 142 160 140 ...
#>  $ heightIn: num  56.3 62.3 63.3 59 62.5 62.5 59 56.5 62 53.8 ...
#>  $ weightLb: num  85 105 108 92 112 ...
```

### 2.2.2 Data Types

Each column (feature) of a data frame is a vector of the same *type* of data. R recognizes many data types, but here are the primary ones we will need to know for data visualization:

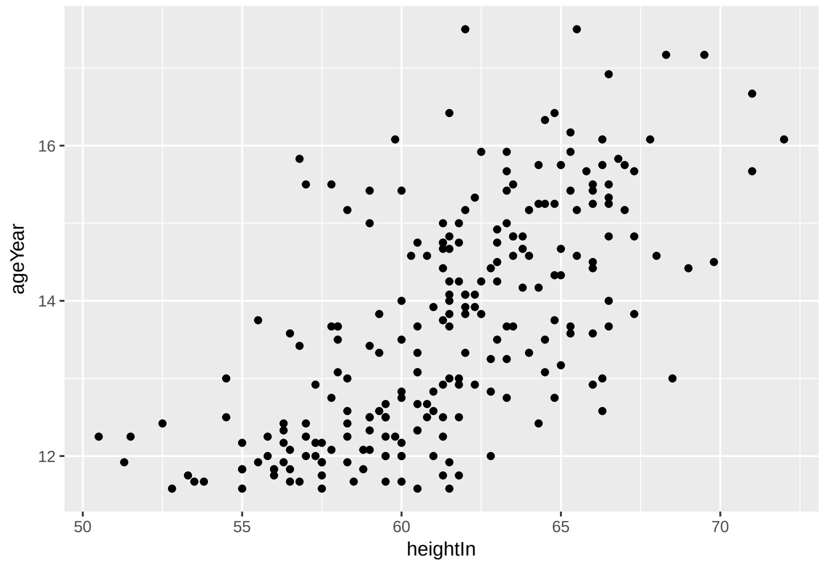
- `numeric` or (`num`) is used for *continuous* variables
- `integer` or (`int`) is used for *integer* variables
  - if an integer column has a few unique values, treat like categorical. Else treat like continuous variable.
- `character` or (`chr`) is used for *categorical* variables
  - ordered alphabetically
- `factor` or (`Factor`) is used for *categorical* variables
  - these are special in that factors also contains the *levels*, or possible values the variable can have.
  - ordered by levels
- `logical` or (`logi`) for *TRUE/FALSE* variables
- `date` or (`Date`) for *date* variables

The data types determine how each variable can be used in a plot. For example, `numeric` variables cannot be used for faceting and `categorical` variables should not be used for the size asthetic. ggplot2 makes the distinction between *discrete* and *continuous* variables on the Data Visualization Cheat Sheet.

## 2.3 Basic Scatterplot

A scatterplot show the relationship between two numeric (continuous) variables. Here is the basic setup with `ggplot2` for examining the relationship between height (`heightIn`) and age (`ageYear`)

```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear))
```



Is is clear that tall children are generally older than shorter children (trend).

**Your Turn #1**

What other patterns or features can you find?

Notice the two components used to build the plot:

1. `ggplot()` initiates a new plot object.
   - `?ggplot`
   - It can take arguments `data=` and `mapping=`.
   - In the example, we used `ggplot(data=heightweight)` making the `heightweight` data available to the other plot layers
2. `geom_point()` adds a layer of points to the plot
   - `?geom_point`
   - It can take several arguments, but the primary one is mapping. The mapping tells ggplot where to put the points.
   - The `x=` and `y=` arguments of `aes()` explain which variables to map to the x and y axes of the graph. ggplot will look for those variables in your data set, `heightweight`.
   - The call `geom_point(mapping = aes(x = heightIn, y = ageYear))` specifies that `heightIn` is mapped to x-axis and `ageYear` is mapped to y-axis.

You complete your graph by adding one or more layers to ggplot(). Here, the function `geom_point()` adds a layer of points to the plot, which creates a scatterplot. ggplot2 comes with other geom functions that you can use as well. Each function creates a different type of layer, and each function takes a mapping argument.

The ggplot components can be on different lines, but must have the + separator **before** the end of line.

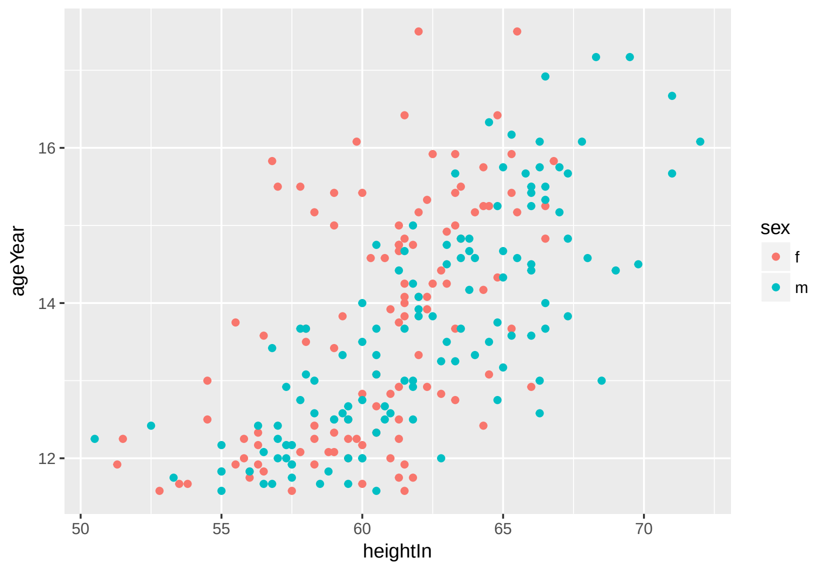
```
#- What is wrong here?
ggplot(data=heightweight)
  + geom_point(mapping = aes(x = heightIn, y = ageYear))
```

## 2.4 Aesthetics

The real strength of ggplot2 is in its mapping of data to a visual component. An *aesthetic* (specified by `aes()`) is a visual property of the points in your plot. Aesthetics include things like the size, the shape, or the color of your points.

It would make sense to examine our data according to `sex` to see if there are differences between the boys and girls. We will use the `color=` aesthetic to color the points according the value of the `sex` variable

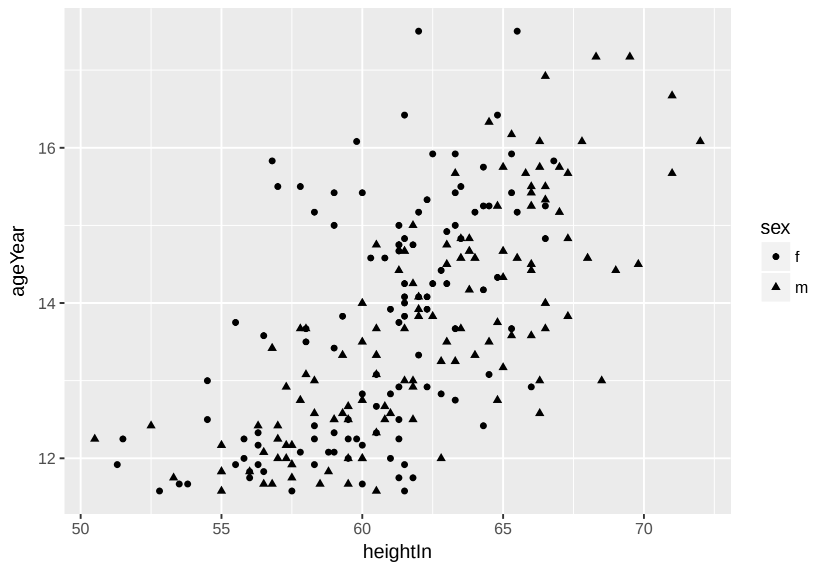
```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear, color=sex))
```



This maps the males (m) point to a blueish color and females (f) to reddish color. (We will illustrate how to change these color mappings later). It also creates a legend that shows the mapping.

We could alternatively try mapping the sex value to a shape (with shape= in aes()):

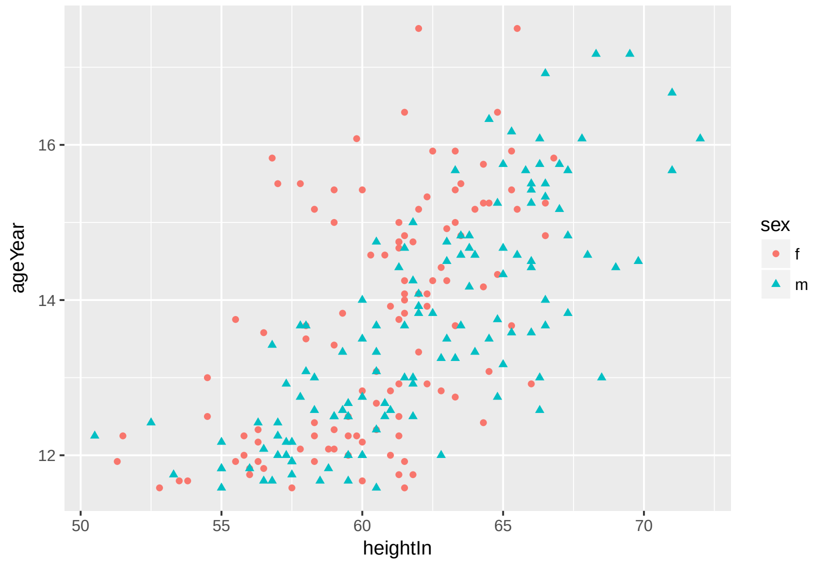
```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear, shape=sex))
```



This, by default, maps the males (m) point a triangle and females (f) to a circle.

We could even map both the color and shape to sex:

```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear, color=sex, shape=sex))
```

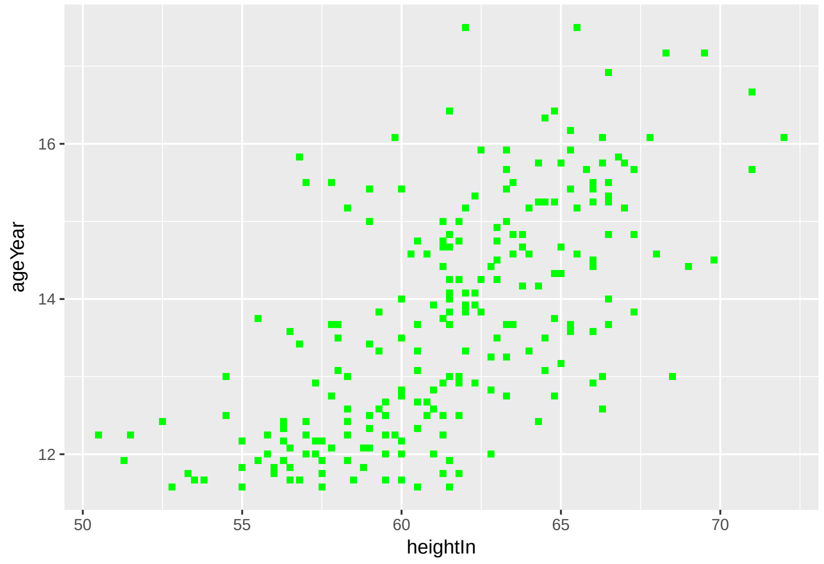


and the legend shows the color and shape.

### 2.4.1 Fixed aesthetics

The previous examples mapped a third variable, `sex`, to the color and shape. But we can also fix these values (not associated with a variable) by setting them outside of `aes()`.

```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear), color="green", shape=15)
```



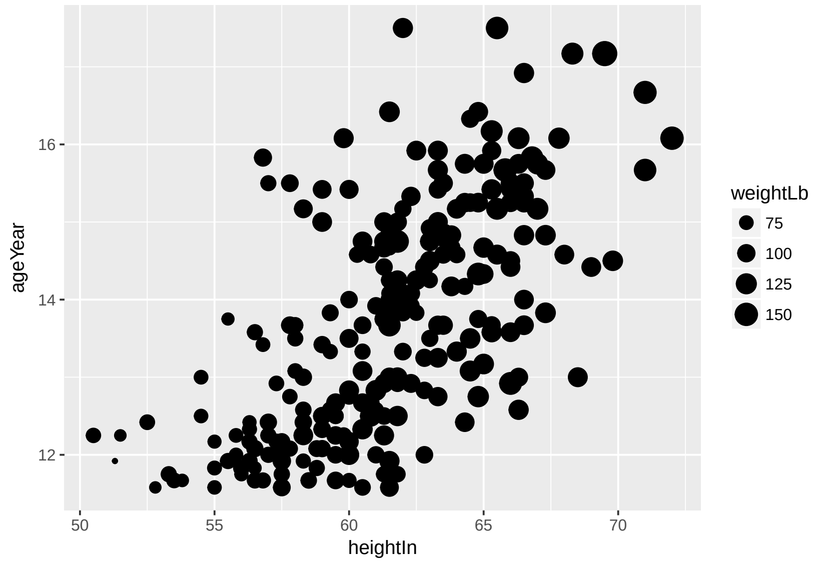
Notice the legend disappears since these are fixed values.

Summary:
- *inside* of the `aes()` function, ggplot2 will map the aesthetic to data values and build a legend.
- *outside* of the `aes()` function, ggplot2 will directly set the aesthetic to your input.

### 2.4.2 Continuous aesthetics

Notice that we mapped *continuous* variables to the x and y axis, and a *discrete* (categorical) variable to the color and shape. We can also map *continuous* variables to the aesthetics. For example, we can make a *bubbleplot* by mapping the size of point to the child's weight (`weightLb`).

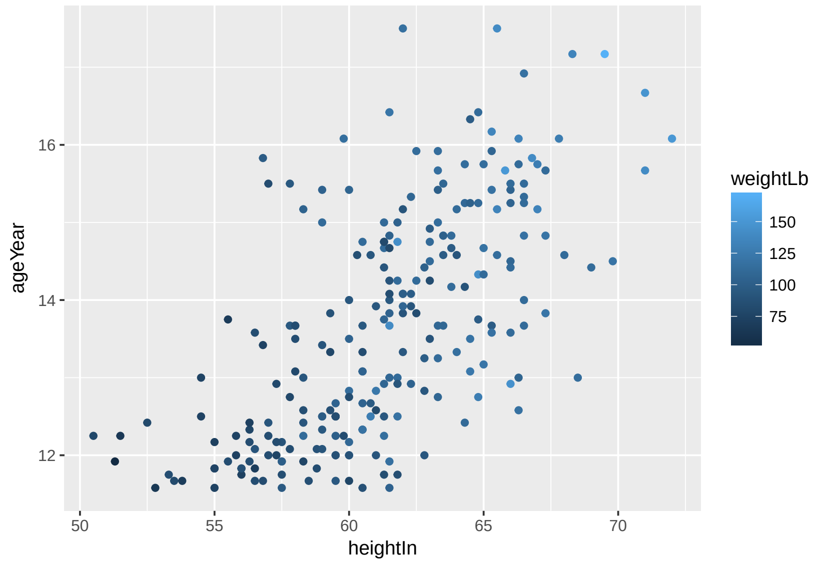
```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear, size=weightLb))
```



The legend shows how the size corresponds to the weight.

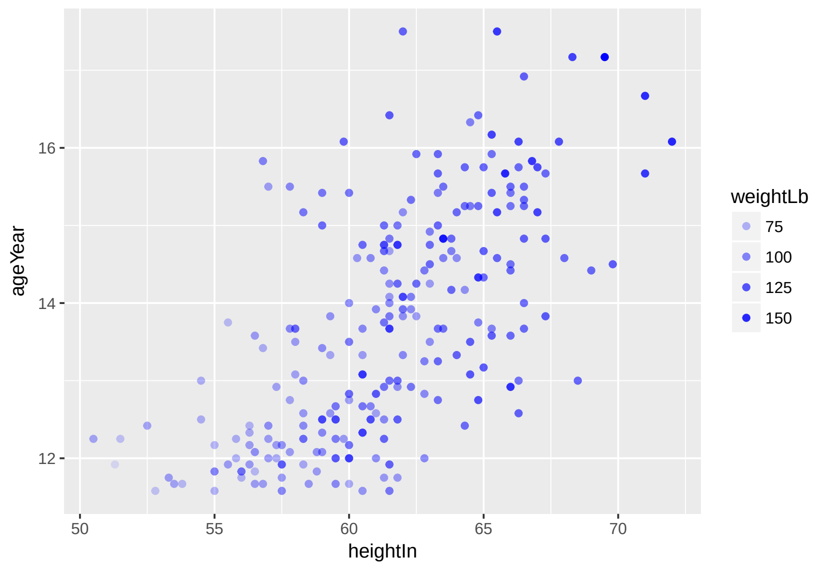
Color can also be set by a continuous variable

```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear, color=weightLb))
```



Similar to color, *alpha* controls the transparency of the color

```
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear, alpha=weightLb),
             color="blue")
```



### 2.4.3 Other aesthetics for `geom_point`

Each geom can understand a set of aesthetics. We can find out what `geom_point()` accepts by checking the help

- `?geom_point`
- http://docs.ggplot2.org/current/geom_point.html
- Data Viz cheatsheet

## 2.5 Your Turn: Scatterplots

**Your Turn #2 : Scatterplots**

Now that you know how to use aesthetics, take a moment to experiment with the `mpg` data set (from `ggplot2 package`).

1. Map a discrete variable to color, size, alpha, and shape. Then map a continuous variable to each. Does ggplot2 behave differently for discrete vs. continuous variables?
   - The discrete variables in mpg are: manufacturer, model, trans, drv, fl, class
   - The continuous variables in mpg are: displ, year, cyl, cty, hwy
2. Map the same variable to multiple aesthetics in the same plot. Does it work? How many legends does ggplot2 create?
3. Attempt to set an aesthetic to something calculated, like `displ < 5`. What happens?

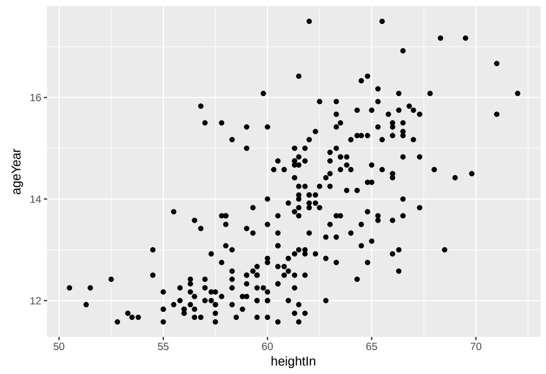
## 2.6 Additional Geoms

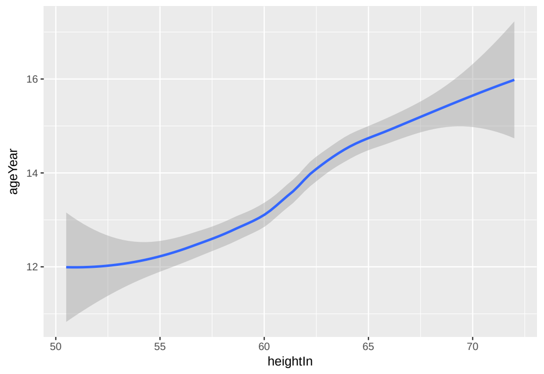
A *geom* is the geometrical object that a plot uses to represent data. People often describe plots by the type of geom that the plot uses. For example, bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, and so on. Scatterplots use the point geom.

You can use different geoms to plot the **same data**. The plot on the left uses the point geom, and the plot on the right uses the smooth geom, a smooth line fitted to the data.

```
#- scatterplot: geom_point()
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear))

#- smoothed curve: geom_smooth()
ggplot(data=heightweight) +
  geom_smooth(mapping = aes(x = heightIn, y = ageYear))
#> `geom_smooth()` using method = 'loess'
```





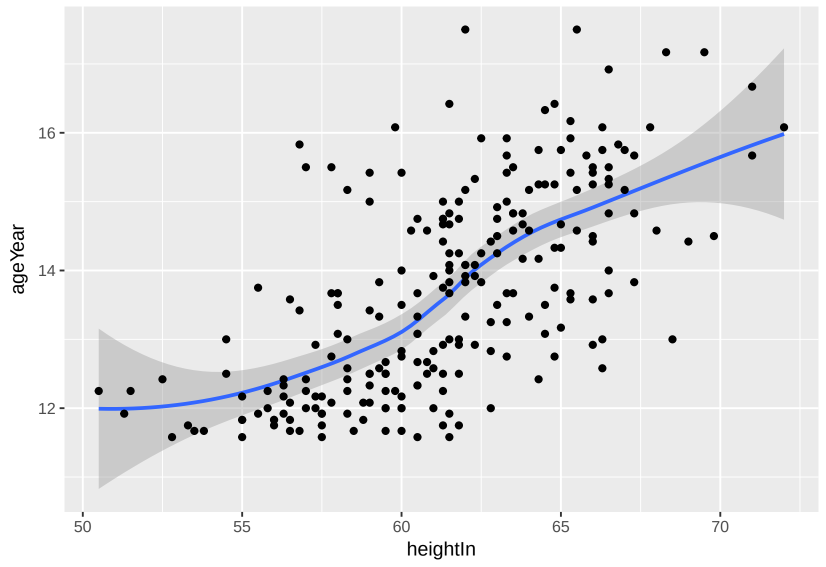
Both plots represent the same data, but in different ways. The smooth curve is an estimate of the mean age for a given value of height (with point-wise 95% confidence interval), while the points are the raw values.

The `geom_smooth()` can take a different set of aesthetics than `geom_point()`.

## 2.7 Layers

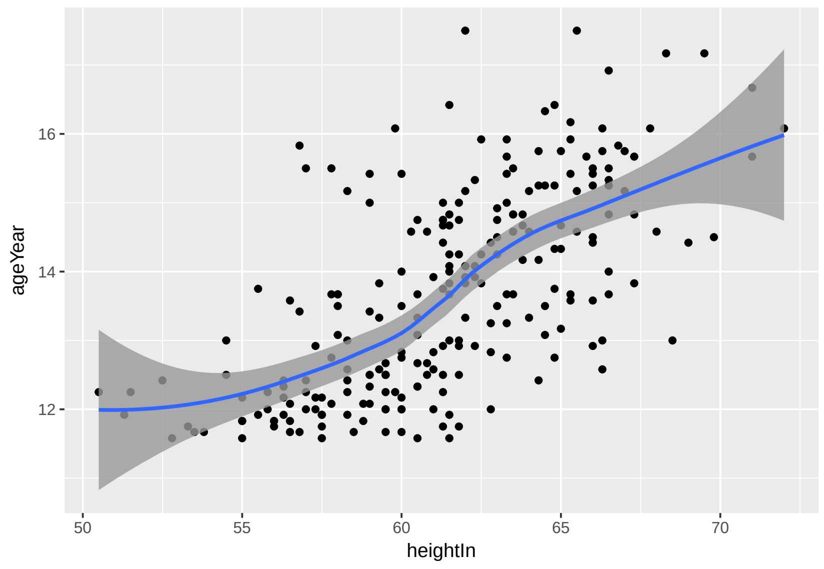
We can also layer geoms to add additional information

```
ggplot(data=heightweight) +
  geom_smooth(mapping = aes(x = heightIn, y = ageYear)) +
    geom_point(mapping = aes(x = heightIn, y = ageYear))
#> `geom_smooth()` using method = 'loess'
```



Note that the order of adding layers matters. If we added the points first (and set a large alpha value), then some of the points would be obscured.

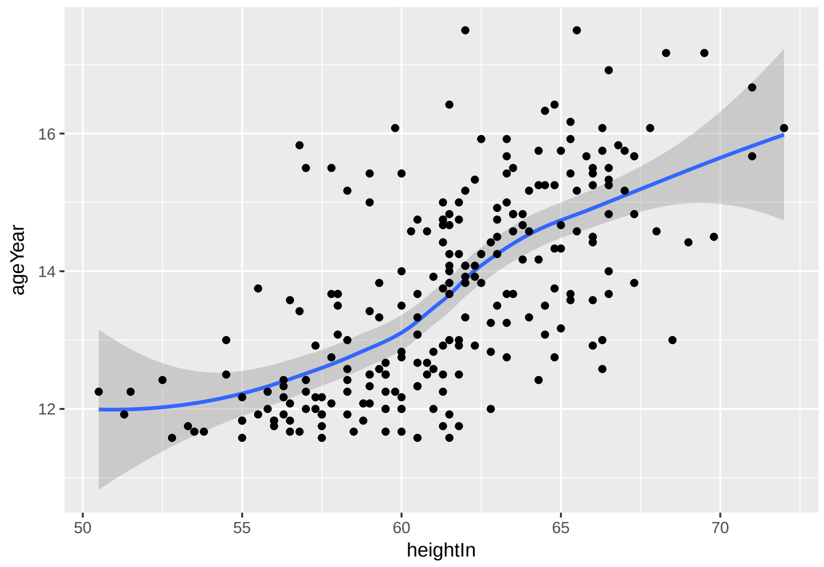
```r
ggplot(data=heightweight) +
  geom_point(mapping = aes(x = heightIn, y = ageYear)) +
    geom_smooth(mapping = aes(x = heightIn, y = ageYear), alpha=.8)
#> `geom_smooth()` using method = 'loess'
```



### 2.7.1 Global vs. local aesthetics

Notice that in the last example, we specified the x and y mappings twice. This is not necessary. *Global* aesthetics can be assigned in the `ggplot()` function and will apply to all layers (unless specified in the geoms). For example, the previous example could be produced by:

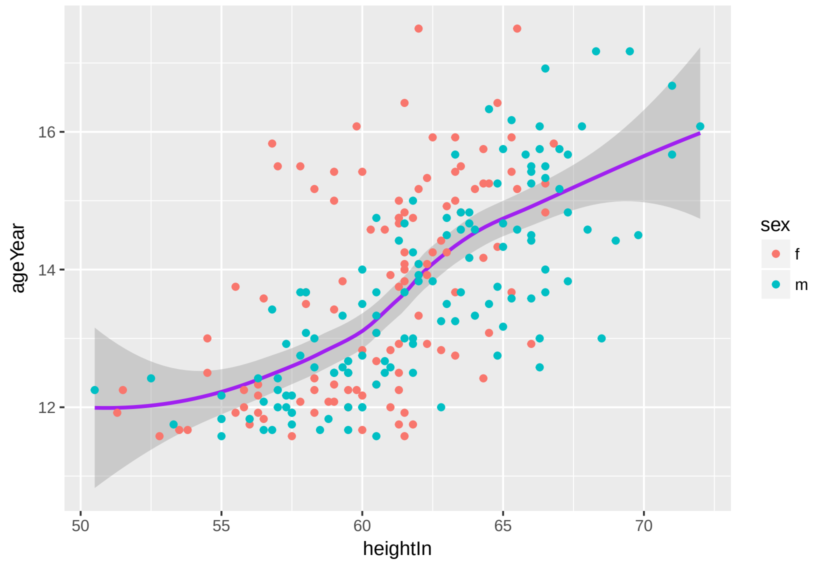
```r
ggplot(data=heightweight, mapping = aes(x = heightIn, y = ageYear)) +
  geom_smooth() +
    geom_point()
#> `geom_smooth()` using method = 'loess'
```



Because the data and x,y mappings are given in `ggplot()`, they apply to both subsequent layers. These global settings can be modified or enhanced at the geom level

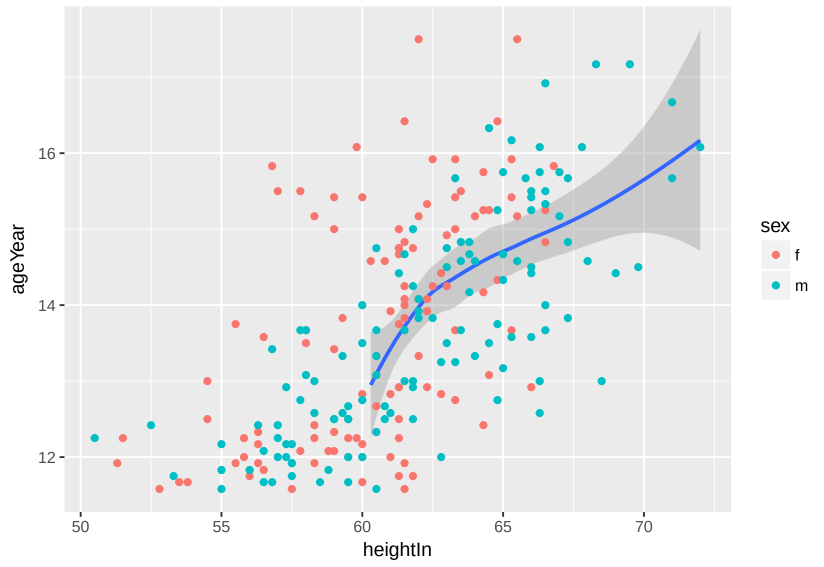
```r
ggplot(data=heightweight, mapping = aes(x = heightIn, y = ageYear)) +
  geom_smooth(color="purple") +                     # change line to purple
```

```
    geom_point(mapping=aes(color=sex))             # map sex to point color
#> `geom_smooth()` using method = 'loess'
```



Also, each geom can use its own data

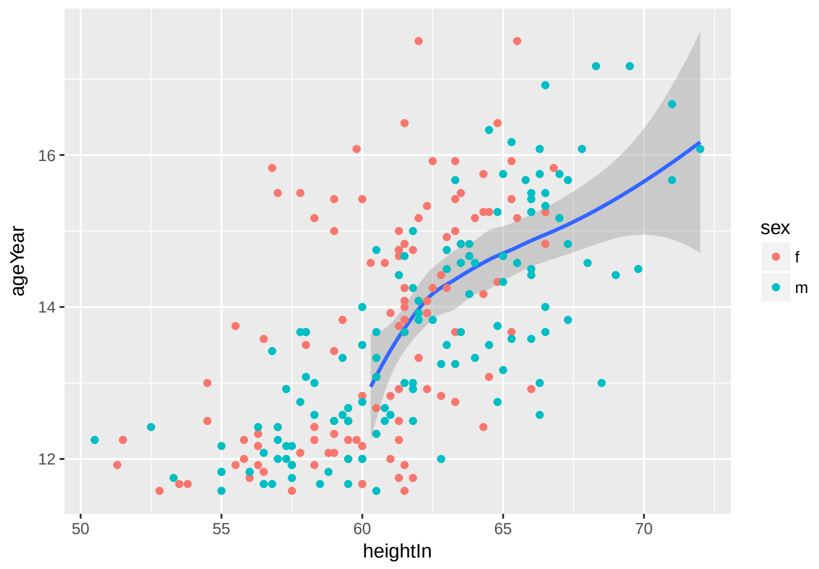
```
ggplot(data=heightweight, mapping = aes(x = heightIn, y = ageYear)) +
  geom_smooth(data=filter(heightweight, heightIn> 60)) + # use filtered data
    geom_point(mapping=aes(color=sex))                   # map sex to point color
#> `geom_smooth()` using method = 'loess'
```



### 2.7.2 Arguments by position

Remember that `ggplot()` and `geom_*()` are functions, so we can set their arguments by name or position. Because `data=` is the first argument of `ggplot()` and `mapping=` is the second, we do not need specify the names, but can use position. Likewise, `mapping=` is the first argument of the geoms. So we can save typing by using position

```
ggplot(heightweight, aes(x = heightIn, y = ageYear)) +
  geom_smooth(data=filter(heightweight, heightIn> 60)) + # use filtered data
    geom_point(aes(color=sex))                           # map sex to point color
#> `geom_smooth()` using method = 'loess'
```



Notice, we still need to name `data=filter(heightweight, heightIn> 60)` in the call the `geom_smooth()` because the first argument is expected to be mapping, not data.

## 2.8 Your Turn: Geoms and Layers

**Your Turn #3 : Geoms and Layers**

1. What does `method='lm'` do for `geom_smooth()`? Try it.
2. What will this produce (describe in words)
```
ggplot(heightweight, aes(x = heightIn, y = weightLb,
                         shape=sex, color=sex)) +
  geom_smooth(aes(fill=sex)) +
  geom_point()
```
3. Why will this not work
```
ggplot(mapping=aes(heightIn)) +
  geom_point(data=heightweight, aes(y=weightLb)) +
  geom_smooth()
```

## 2.9 Plot Objects

Every ggplot function returns a plot object that can be saved in the environment and reused as necessary. For example

```
g = ggplot(heightweight, aes(x = heightIn, y = ageYear))
```

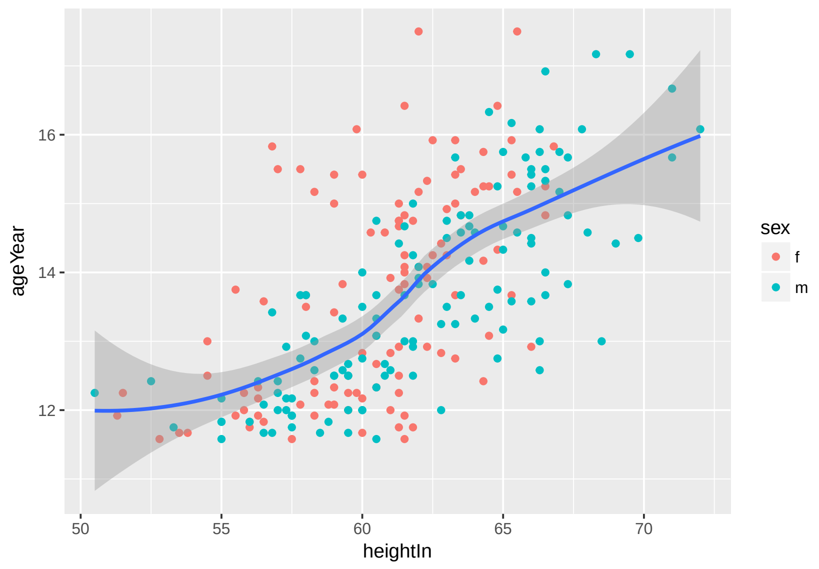
Now go to the environment tab and see the structure of `g`. It is a list with all the specifications for plotting, along with the data. But so far no layers.

We can add a layer, but not plot it yet

```
g2 = g + geom_point(aes(color=sex))
```

To make the plot, we have to explicitly print it.

```
g2 + geom_smooth()   #-or explicitly- print(g2 + geom_smooth())
#> `geom_smooth()` using method = 'loess'
```



This makes is easy to save plots, or components, for reuse and reduce possibility of introducing errors.

Plots can be saved with the `ggsave()` function. For example

```
ggsave("figs/myplot.pdf", plot=g2)
```

**Your Turn #4 : Plot Objects**

1. Where would this plot be saved on your machine?
2. Describe the plot that will be saved?

## 2.10 Scatterplot Aesthetics

Scatterplots can take several aesthetics: `x`, `y`, `alpha`, `color`, `fill`, `shape`, `size`, and `stroke`. These options can be found
- `?geom_point`
- Data Viz Cheatsheet

Some detail of the aesthetics can be found in the ggplot2 vignettes: http://docs.ggplot2.org/current/vignettes/ggplot2-specs.html.

Mapping variable values to colors from cookbook-r

- The `alpha=` aesthetic controls the transparency
- `alpha` takes values in $[0, 1]$.
  - 1 (default value) for no transparency
  - 0 is fully transparent

**Your Turn #5 : Aesthetic Mapping**

1. What is the difference between setting an aesthetic and mapping an aesthetic to a variable?
2. Explain how variables are mapped to aesthetics.

## 3 Bar Graphs: `geom_bar()`

### 3.1 `diamonds` data

The ggplot2 package provides the `diamonds` data, which contains the prices and other attributes of over 50K round cut diamonds. (Type `?diamonds` for details.)

```
data(diamonds)  # load the diamonds data from ggplot2 package
diamonds        # print the diamonds data frame
#> # A tibble: 53,940 × 10
#>     carat       cut color clarity depth table price     x     y     z
#>     <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1    0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
#> 2    0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
#> 3    0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
#> 4    0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
#> 5    0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
#> 6    0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
#> 7    0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
#> 8    0.26 Very Good     H     SI1  61.9    55   337  4.07  4.11  2.53
#> 9    0.22      Fair     E     VS2  65.1    61   337  3.87  3.78  2.49
#> 10   0.23 Very Good     H     VS1  59.4    61   338  4.00  4.05  2.39
#> # ... with 53,930 more rows
```

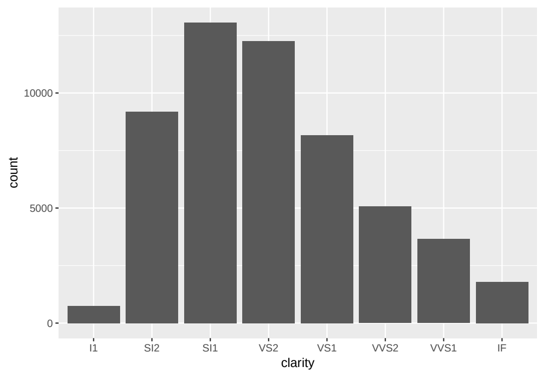
### 3.2 Bar graphs

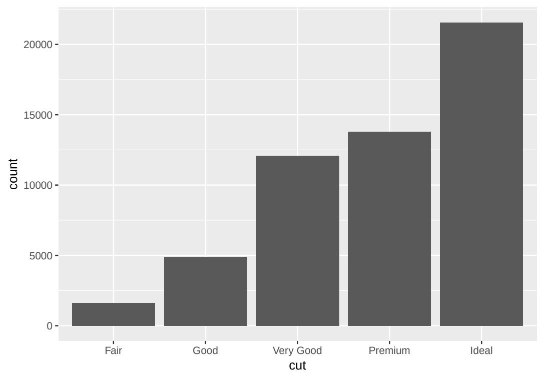
Bar graphs are used to display numeric values (on the y-axis) for different *discrete values* (on the x-axis). A (usually filled) bar extends from 0 to the numeric value. This is a common type of plot to visualize frequency tables (counts of cases in the data).

### 3.3 `geom_bar()`

The basic analysis of categorical/discrete data is a count of how many times each value occurs. This is achieved with the basic frequency table and corresponding bar graph. By default, `geom_bar()` will count the occurrences *and* produce a bar graph.

```
g2 = ggplot(data=diamonds)        # set-up the basic plot object
g2 + geom_bar(aes(x=clarity))     # bar plot of clarity variable
g2 + geom_bar(aes(x=cut))         # bar plot of cut variable
```





Notice that only the x axis must be specified as ggplot2 will *calculate the counts automatically*.

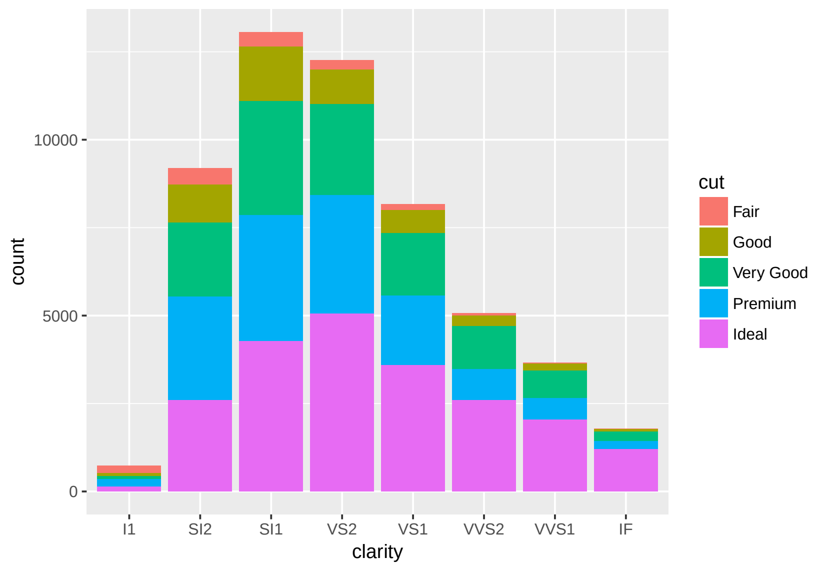
## 3.4 Two Variables

What is more interesting is to examine the joint occurrence of two discrete variables (e.g., `clarity` and `cut`). We will examine four ways to do this: stacked bar graph, side-by-side bar graph, count plot, and faceting.

### 3.4.1 Stacked bar graph

Here we will use `clarity` as the x-axis and map `fill=` to the `cut` variable to make a *stacked* bar graph

```
g2 + geom_bar(aes(x=clarity, fill=cut))
```

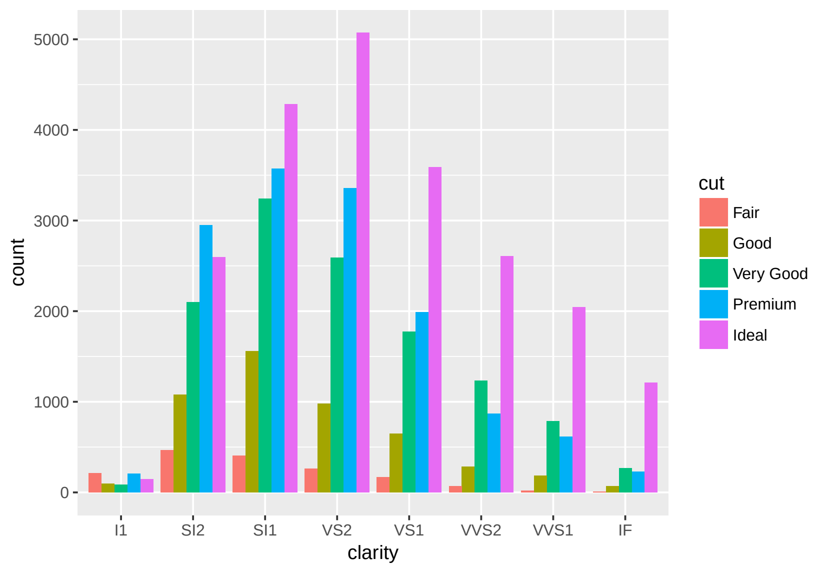


Why did we specify `fill=` instead of `color=`?

### 3.4.2 Side-by-Side bar graph

Another option is a *side-by-side* bar graph. This is achieved by setting `position="dodge"`

```
g2 + geom_bar(aes(x=clarity, fill=cut), position="dodge")
```

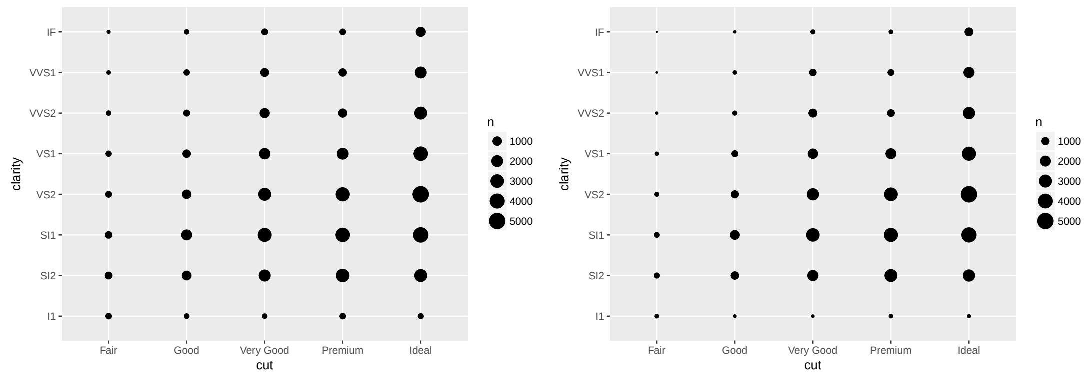


Notice in the help (`?geom_bar`), the default `position="stack"`.

### 3.4.3 Bivariate Counts

The *count plot* produces a two-dimensional distribution.

```
g2 + geom_count(aes(x=cut, y=clarity))
g2 + geom_count(aes(x=cut, y=clarity)) + scale_size_area()
```

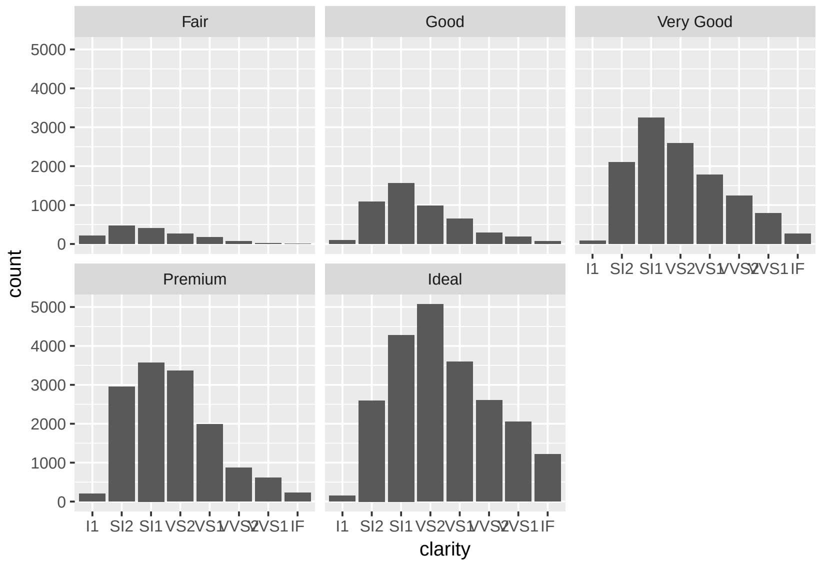


The *area* of the point is proportional to the counts, but zero counts can still get an area. Use the `scale_size_area()` function to ensure that a count of 0 is mapped to a size of 0 (i.e., no dot).

### 3.4.4 Facets

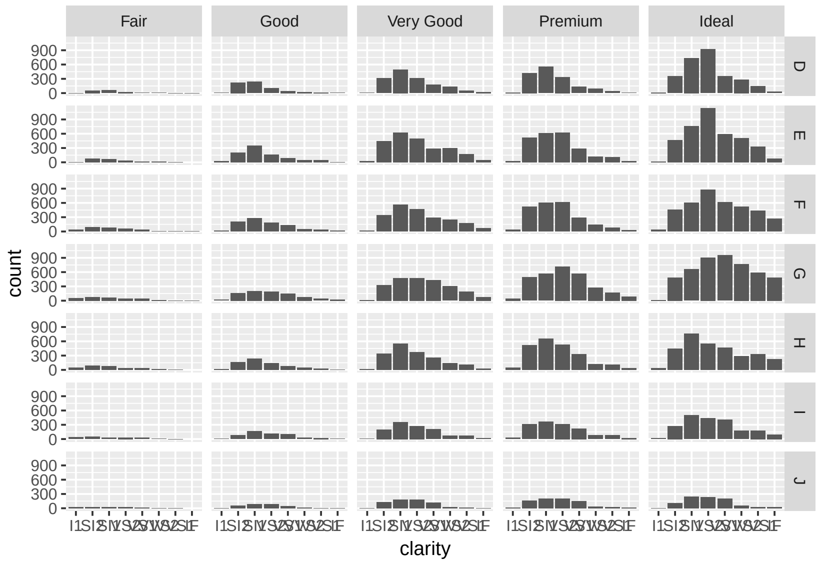
Another way to show the barplot for two (or three) variables is with faceting.

```
g2 + geom_bar(aes(x=clarity)) +
  facet_wrap(~cut)
```



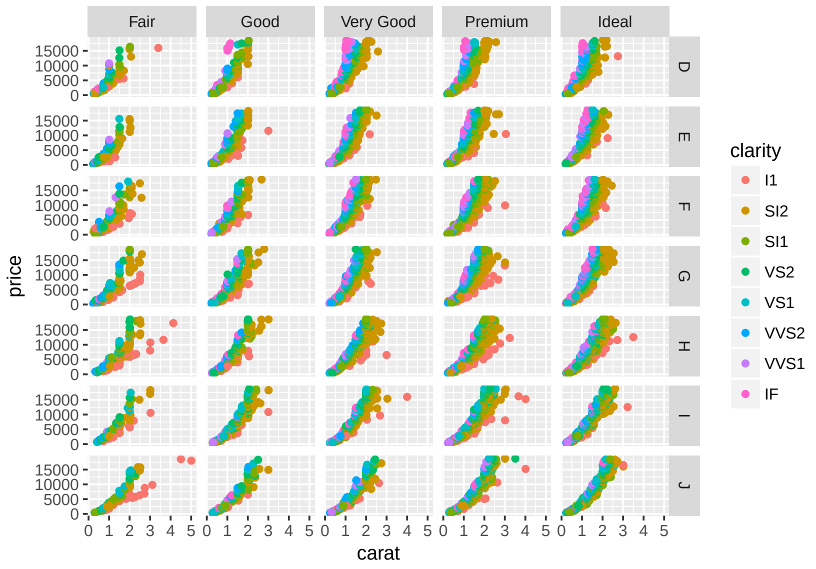
The `facet_wrap()` function will display the plot over a wrapped grid. While `facet_grid()` will display in a grid and is most suitable for two variable faceting

```
g2 + geom_bar(aes(x=clarity)) +
  facet_grid(color~cut)
```



Note that faceting will work with other geoms. This faceted scatterplot shows <u>five</u> variables.

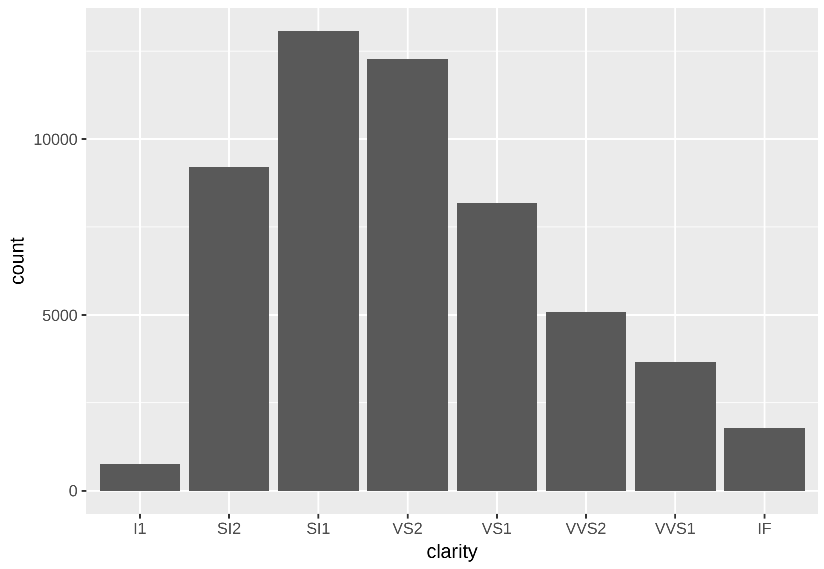
```
ggplot(diamonds) + geom_point(aes(x=carat, y=price, color=clarity)) +
  facet_grid(color~cut)
```



## 3.5 Stats: `stat_count()` and `stat_identity()`

Notice in the basic bar graph:

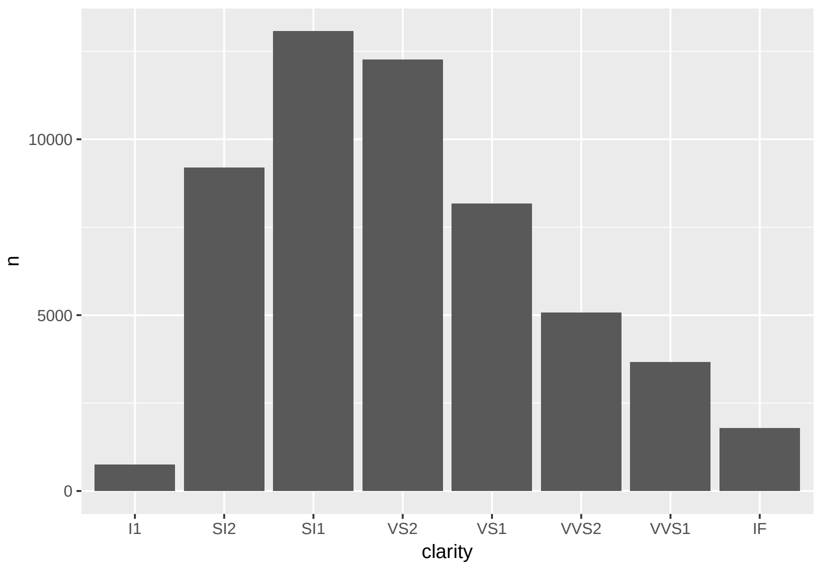
```
g2 + geom_bar(aes(x=clarity))
```



we only set the x-axis aesthetic and ggplot knew to calculate the frequency and assign the y value the counts (with label). It did this by creating new data with a with a **stat**, or statistical transformation, and then applying the geom. Specifically, `geom_bar()`, by default, uses `stat_count` which computes a data set of counts for each x value from your raw data. geom_bar() then uses this computed data to make the plot.

We could mimic this by first creating the counts data, but then we would need to use `stat=identity` so no further transformations are used.

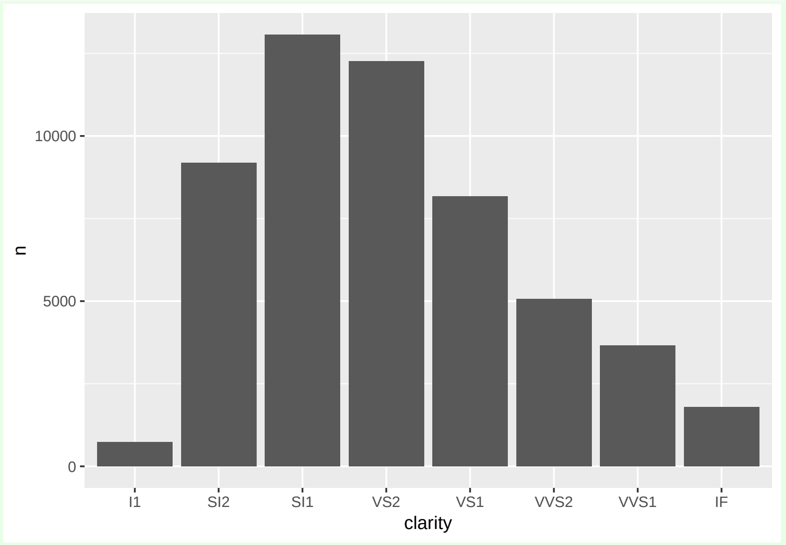
```
d.table = count(diamonds, clarity)      # count() returns the frequencies
d.table
#> # A tibble: 8 × 2
#>   clarity     n
#>     <ord> <int>
#> 1      I1   741
#> 2     SI2  9194
#> 3     SI1 13065
#> 4     VS2 12258
#> 5     VS1  8171
#> 6    VVS2  5066
#> 7    VVS1  3655
#> 8      IF  1790
ggplot(data=d.table) + geom_bar(aes(x=clarity, y=n), stat='identity')
```



More details on the *stats* are given in: http://r4ds.had.co.nz/data-visualisation.html#statistical-transformations.

The new `geom_col()` function is shortcut for `geom_bar(..., stat='identity')`

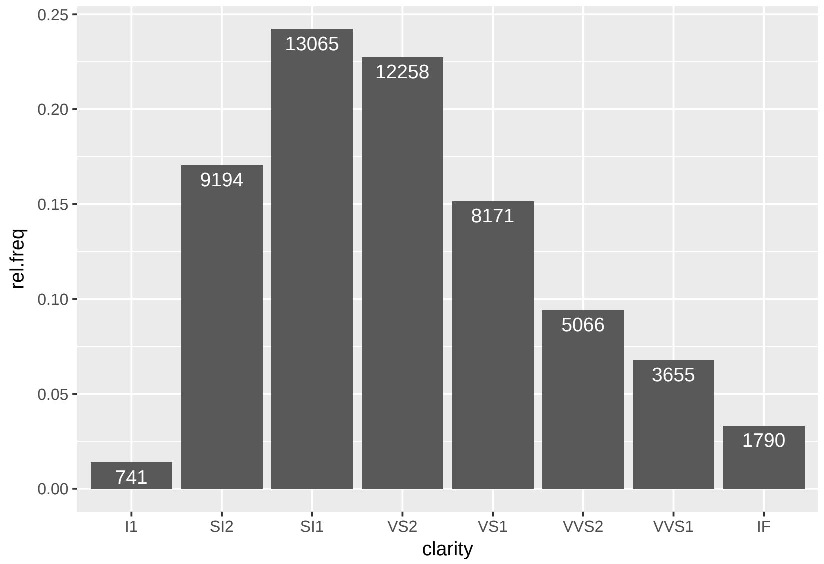
```
ggplot(d.table) + geom_col(aes(x=clarity, y=n))
```



### 3.5.1 Relative frequencies

I often work with relative frequencies rather than the raw counts. It is not difficult to pre-calculate our relative frequencies

```
d2 = mutate(d.table, rel.freq=n/sum(n))
d2
#> # A tibble: 8 × 3
#>   clarity     n rel.freq
#>     <ord> <int>    <dbl>
#> 1      I1   741  0.01374
#> 2     SI2  9194  0.17045
#> 3     SI1 13065  0.24221
#> 4     VS2 12258  0.22725
#> 5     VS1  8171  0.15148
#> 6     VVS2   5066   0.09392
#> 7     VVS1   3655   0.06776
#> 8       IF   1790   0.03319
ggplot(data=d2) +
         geom_bar(aes(x=clarity, y=rel.freq), stat='identity') +
         geom_text(aes(x=clarity, y=rel.freq, label=n), vjust=1.5, color="white", size=4)
```



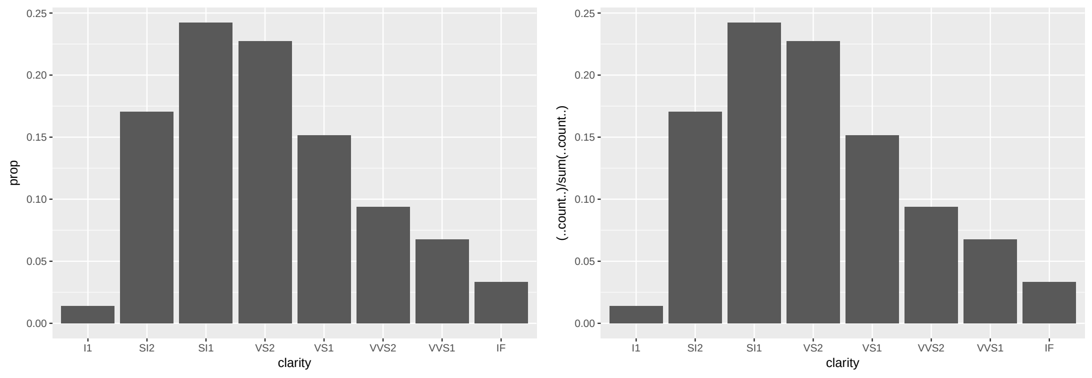
Notice how the `geom_text()` is used with the `label=` aesthetic.

The `mutate()` functions adds a calculated column to a data frame. We will learn about this function later.

Alternatively, we can use the output (i.e., computed variables) from the `stat_count`, which is ...count... and ...prop...

```
#-- Alternative Relative Frequency Bar Plots
g2 + geom_bar(aes(x=clarity, y=..prop.., group=1))

g2 + geom_bar(aes(x=clarity, y = (..count..)/sum(..count..)))
```



## 3.6 Reordering x-axis `reorder()`

How is the x-axis ordered? If the variable mapped to the x-axis is:

- numeric (or integer), then it orders from smallest to largest
- factor, then it orders according to the levels

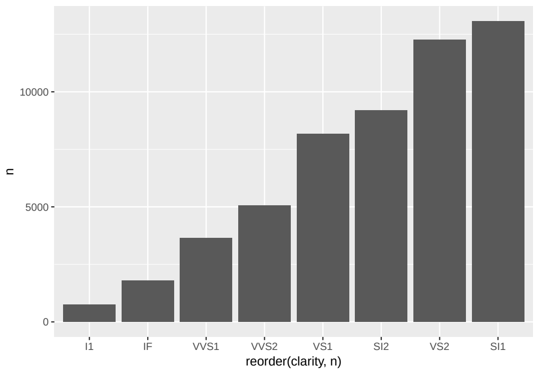
- character, then it orders alphabetically (basically converts to factor)

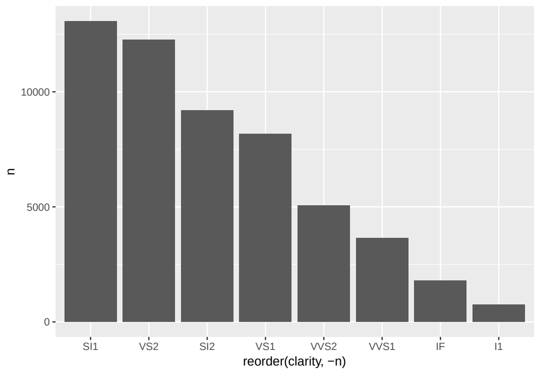
But we are creating graphics to reveal something about the data. And the ordering of the bars can bring understanding. There are two primary orderings: inherent ordering of the levels or height of the bars.

In the diamonds data, the factors (`cut`, `color`, and `clarity`) are already ordered from worst to best quality. But we may want to reorder from the most frequent level. This can be achieved with the `reorder()` function.

```r
#- increasing order
ggplot(data=d.table) + geom_bar(aes(x=reorder(clarity, n), y=n), stat='identity')

#- decreasing order
ggplot(data=d.table) + geom_bar(aes(x=reorder(clarity, -n), y=n), stat='identity')
```





### 3.7 Your Turn: Bar Graphs

**Your Turn #6 : Bar Graphs**

Using the `mpg` data from `ggplot2` package:

1. Make a bar graph of `cyl` and facet by `drv`
2. Make a side-by-side bar graph of `cyl` with a `fill=` according to `year`

## 4 Additional Material

### 4.1 ggplot 2 details

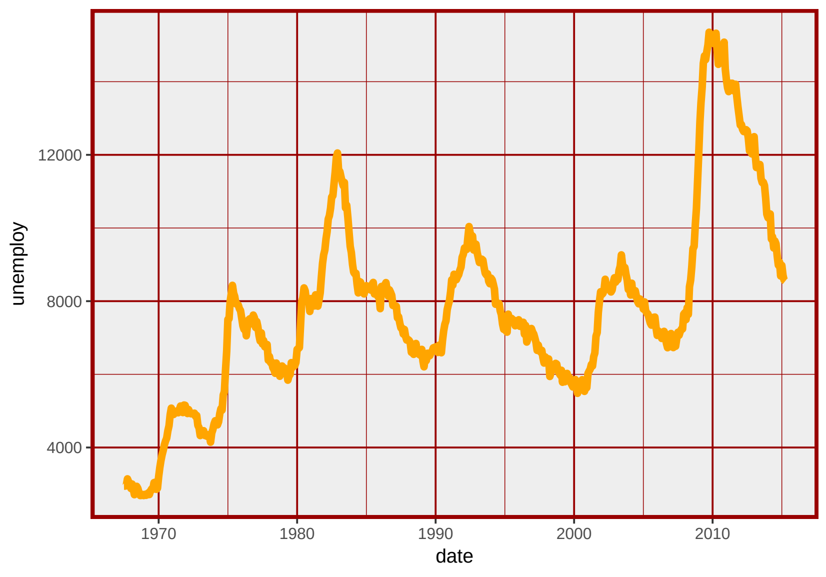
- **data** is what we want to visualize. Consists of variables, which are the columns of a data frame (data must be a data frame)
- **geoms** are geometric objects that are drawn to represent data (bars, lines, points, etc.)
- **aesthetics** are the visual properties of geoms, such as x and y position, line color, point shapes, etc.
- **mappings** from data values to aesthetics (map a factor value to a color)
- **scales** control the mapping from the values in data space to values in aesthetic space. E.g., a continuous y scale maps larger numeric values to vertically higher positions in space.
- **guides** show the viewer how to map visual properties back to data space (e.g., tick marks, axis labels, etc.)

## 4.2 Themes

Alabama Theme, using the official UA colors. See RGraphics Cookbook Chapter 9 for more details.

```r
theme_UA = theme(
      panel.background = element_rect(fill='#eeeeee'),
      panel.grid.major = element_line(color="#990000"),
      panel.grid.minor = element_line(color="#990000",  size=0.2),
      panel.border = element_rect(color="#990000", fill=NA, size=2))

ggplot(economics, aes(date, unemploy)) +
  geom_line(size=2, color="orange") +
  theme_UA
```



## 4.3 Scales

The *scale* controls the mapping from the values in data space to values in aesthetic space. We have already used a few scale adjustments. `scale_size_area()` sets the size of a point proportional to the area, not radius. `scale_fill_brewer()` and `scale_color_brewer()` change the fill and color mappings according to colorbrewer palettes. You can also change the coordinate scale (`scale_x_log10`), etc. Type `scale_` and hit Tab from RStudio to scroll through all options and http://docs.ggplot2.org/current/ and the R Graphics Cookbook for specific examples and options.